UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

THE SCIENCE OF FREE/OPEN SERVICES

Vincenzo D'Andrea, G.R.Gangadharan

August 2006

Technical Report # DIT-06-052

The Science of Free/Open Services

Vincenzo D'Andrea & G.R.Gangadharan

Department of Information and Communication Technology,
University of Trento,
Via Sommarive, 14, Trento, 38050 Italy

Abstract. With their inherent potentiality, services proliferate in myriad domains, though service oriented computing is in incipient nature. Free/Open Source Software is an established approach serving society through technology in an interdisciplinary way. Inspired by Free/Open Source methodology, in this paper, we describe the concept of Free/Open service and analyze the implications of Free/Open source approach on the different aspects of service oriented computing.

Of every tree of the garden thou mayest freely eat; but of the tree of the knowledge of good and evil, thou shalt not eat of it. - Genesis: Holy Bible

1 Introduction

Even an apple is available freely on your way, you have to consider the issues of property rights before the consumption of it. If, for an apple, considering the rights of consumption is significant, then for software which is freely available, it becomes inevitable to consider the rights of consumption. The Free/Open Source Software (FOSS) approach protects the unconditional rights of modification and redistribution by the collaborating developers, making the source code freely available [1]. The freedom in software is reflected by the software license describing terms and conditions for use and distribution.

Service oriented computing (SOC) is an emerging distributed systems paradigm, addressing the aspects of real world applications, crossing organizational and technical boundaries. With a vision of dynamically composing service oriented and non-service oriented applications, SOC continues to penetrate as a technology for connecting applications in a loosely coupled manner. Though web services proliferate seamlessly, many available web services are not even considered as providing relevant business value.

Adopting and adapting the principles of FOSS approach to SOC could enhance the widespread use of services. We introduce a novel concept of Free/Open services (F/O-Services), inspired by FOSS movement over SOC.

The rest of the paper is organized as follows. Section 2 presents free / open source software concepts. Section 3 introduces the concept of service oriented computing. A comprehensive description of what we mean by F/O-Services is elucidated in Section 4. Section 5 elaborates F/O-Services, exemplifying the freedom and openness given by a license. The business models for F/O-Services are depicted in Section 6. Section 7

presents FOSS development patterns for service implementation. Section 8 describes the consequences of adoption of FOSS principles in services paradigm. Section 9 discusses related work, and Section 10 draws some conclusions and future directions of research.

2 Free / Open Source Software

Free / Open Source Software (FOSS) is a family of software allowing the users to explore, modify, and improve the software through the availability of source code. FOSS is an encompassing term for the development models, the legal terms, and the sociological issues associated to the novel software paradigm.

Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. According to [2], it refers to four kinds of freedom, for the users of the software:

1. The freedom to run the program, for any purpose (freedom 0).
2. The freedom to study how the program works, and adapting to the needs (freedom 1).
3. The freedom to redistribute copies (freedom 2).
4. The freedom to improve the program, and release improvements to the public, so that the whole community benefits (freedom 3).

Open source software, as a superset of free software, exists in a plethora of initiatives today, representing a variety of technology innovations and approaches [3]. Some of the key conditions of Open Source Definition (for authoritative definition, see [4]) are as follows:

- The software should be freely redistributable.
- The software must include source code, and must allow distribution in source code as well as compiled form.
- The software must permit modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
- The rights attached to the software must apply to all to whom the software is redistributed without the need for execution of an additional license.
- The license must not discriminate against any person or group of persons or any field of endeavor.

Even though there are fundamental differences between Free Software and Open Source Software, we refer them together by a single term 'FOSS'.

3 Service Orientation of Software

Most of the products fall in a continuum having pure service on a terminal point and pure commodity good on the other one [5]. Software, traditionally, has been perceived as a product, requiring possession and ownership, in order to receive the desired performance. Software-as-a-service [6] is a mechanism of renting software where users

are subscribed to the software they use. SOC allows the software-as-a-service concept to expand to include the delivery of complex business process and transactions as a service, allowing applications to be constructed on the fly and services to be reused everywhere and by anybody [7].

The idea of software composition and refinement instead of software development from scratch nowadays is elaborated to the platform-independent, distributed and standardized services paradigm [8]. In such paradigm, services reflect self-contained processes that can be described, published, discovered and invoked in a distributed environment, connecting people, processes, and applications. Services are intended to represent meaningful business functionality that can be federated with other services, to enhance more value to the business functionality.

Service provider exposes the business functionality in the form of service. Service provider, in other words, the owner of web services, decides the functions to be exposed, negotiation, and pricing strategies. A service is advertised in a public registry through ***Publish*** operation. Service requester can be either a consumer consuming services directly or a provider acting as an aggregator of services. Ultimately, service requester is the user for the published services. A service requester communicates with service broker through ***Find*** operation to select the most appropriate service to satisfy specific requirements. Further, the Service requester interacts with the concerned service provider through ***Bind*** operation and uses the service. Service broker is a registry where the descriptions of the services are stored. Based on the information in the registry about a service, service requester contacts the corresponding service provider and thus consumes the service.

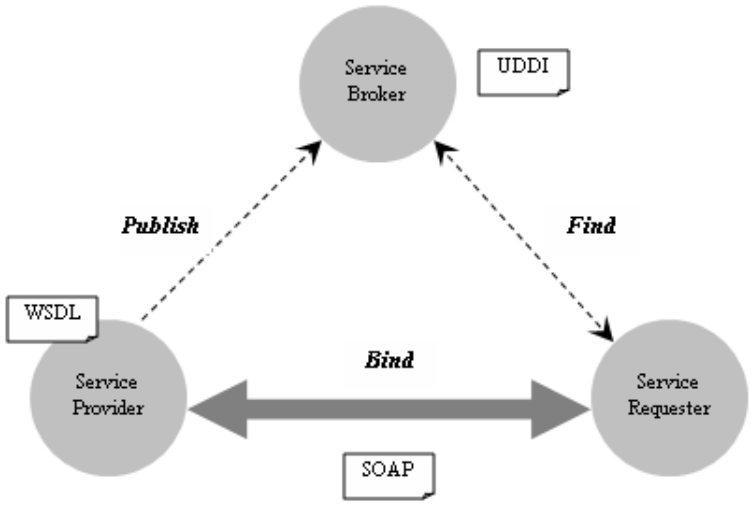

Fig. 1. Service Oriented Computing (Instances with Web Service)

A service is represented by an interface part defining the functionality visible to the external world as a means to access the functionality and an implementation part realizing the interface. The interface part is the description of the service having all the specifications for invoking the service. The service description contains the operations (like a method signature in a programming language), protocol, data formats, specifying how a service interface is implemented by the service provider. The implementation part is the realization of the interface. A service implementation could provide the functionality directly or could combine other services to provide the same functionality. A service can be implemented in any language.

The application of SOC model (see Figure 1) to web resources is manifested by web services to provide a loosely coupled model for distributed processing. Web services are the enabling technology, standardized to construct and integrate applications and organizational interfaces as services, using the Internet as the communication medium and open Internet-based standards [9]. The Web Services Definition Language (WSDL) is an XML based interface definition language, describing services as a collection of messages (abstract descriptions of the data being exchanged) and port types (abstract collections of operations), separated from their concrete network deployment or data format bindings. Universal Description, Discovery, and Integration (UDDI) enables publishing and accessing WSDL specifications in directories. Simple Object Access Protocol (SOAP) is a platform and language independent protocol, providing a way of communication between applications.

Service composition [10] is related to the implementation of a web service whose internal logic involves the invocation of operations offered by other web services. Services can be composed as a part of composite service, encapsulating (see Figure 2a) individual services and exposing a different set of operations. Another perspective on composition is by defining the invocation order (see Figure 2b) of individual services [11]. Service composition allows a recursive process of composition of services i.e. a composed service can be composed with an other elementary and/or composite service. Thus, individual services can be composed up to any levels of hierarchies.

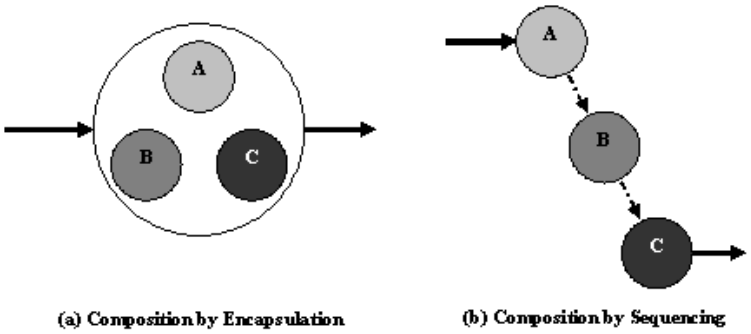

Fig. 2. Service Composition

Though the concept of arbitrarily mixing and matching the services from different providers seems interesting, the basic clauses of service licenses would enforce certain terms and conditions on composition. Questions of ownership and distribution could impede composition, thereby impacting the reuse of services. To illustrate the issues that could arise in the context of licensing web services, we consider a simple scenario (see Figure 3) where R is a restaurant service providing the following operations: R_0, information on location and opening hours; R_1, the facility for reserving table; R_2, a catalogue of specialty cuisines; R_3, a daily recipe for one of the specialty cuisine. Another service, F, a restaurant finder service uses R, for the following operations: F_1, a restaurant locator giving a list of restaurants close to a given location and using R_0 (as well as similar operations for other restaurants); F_2, for intermediating table reservation, using R_1; F_3, a daily recipe randomly selected among the recipes provided by the restaurants listed using F (in the case of R, it will use operation R_3). The license terms of R may deny the provision of R_3 to other services intended for providing recipe information exclusively or may require attribution for the use of R_3. The license terms of R can even require the same set of terms and conditions for any hierarchy of composed services, even the successive compositions use F. In this case, the license terms of F will have to comply with R, for the request and deny provision of F_3 to other services intended to provide the recipe information exclusively. Another restaurant service, S, has a similar set of operations S_0, S_1, S_2, S_3 as R, but having a different license that freely allows the use of operations anywhere. If F uses also S, then it could be possible to have a different license when F_3 presents a recipe chosen from S. Even in this simple scenario, it is apparent that the composition of licenses could easily bring to incompatibility between the composed services.

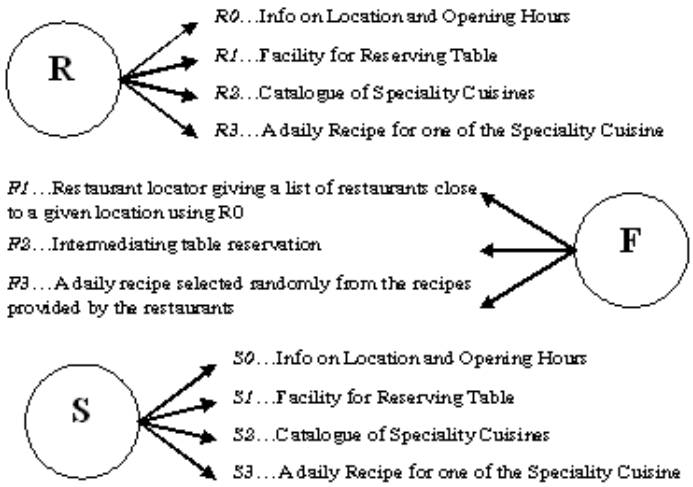

Fig. 3. Service Composition Illustration

Besides the functional operations, from the point of view of a service consumer, it is important to consider also other, non-functional, aspects of service provisioning, such as the cost or the reliability of a service. These aspects are collectively referred to as Quality of Service (QoS) or non-functional properties of a service. The QoS of a composite service is derived from the aggregation of QoS of each individual services, where the aggregation could be a simple combination such as adding the cost of individual services, or taking the maximum among the performances of the individual services to estimate the response time of a composite service. For other aspects, the combination requires the definition of a specific model, such as combining security aspects or reliability, availability, scalability and so on.

4 Free/Open Services (F/O-Services)

Following FOSS definitions [2, 4], we define a F/O-Services as follows:

1. A F/O-Service should be free for use.
2. The source code of the interface (WSDL descriptions) as well as the implementation of a F/O-Service should be available.
3. The service implemented by creating a new service using the source code and interface of a F/O-Service should be freely distributable as an independent service. The modification of interface and implementation should be permitted.
4. The service using a F/O-Service as part of a composite service should be freely distributable as an independent service, even when using a separate interface. The modification of interface and implementation should be permitted.
5. Derived services and modified services must be allowed and be capable of distribution.
6. The license must not discriminate against any person or group of persons or any field of endeavor.
7. The license agreement must provide a F/O-Service “as is” with no warranties either to functional and/or non-functional properties or non-infringement of third party rights.
8. The license must not place restrictions on composition with other services and on distribution of composed services.

Thus, a F/O-Service enhances the way of usage, adaptation, distribution, and redistribution of a service as follows:

1. ***Service Usage***
   Service usage describes the freedom to execute a service by other applications, for any purpose. The basics of F/O-Service allows the use (execution) of service by any other service oriented and/or non-service oriented applications, adhering the given F/O-Service license.
2. ***Service Implementation***
   With the creation of F/O-service, we are provided with the freedom to know how the service works and could be adapted to our needs, making the source code of service interface as well as service implementation freely available.

(a) A service is described by WSDL. Service orientation obligates WSDL code to be available publicly for service discovery, and composition.
(b) In addition, a F/O-Service allows the availability of the source code of implementation (the real functionality of a service).
(c) The source code of a service wrapping the functionality of another proprietary software partially or fully, can be available publicly with service interface and implementation, except the source code of proprietary software being wrapped in the given service.

3. ***Service Distribution***
Service redistribution describes the freedom to distribute a service as a separate service. Further, any entity can create a new service which would use the interface of a F/O-Service, without the need to implementing the service realization.
4. ***Service Derivation & Redistribution***
Service derivation and distribution offer the freedom to improve the service, and release improvements to the public, so that the whole community benefits. F/O-Services allow to perform modifications on the WSDL interface and implementation of the service and thus, derived services are created. Derived services could be executed independently (together with separate interface and implementation) or could use the implementation of the parent service.

5 Exploring the Freedom and Openness in F/O-Services

A F/O-Service allows free execution of the service with other applications, making the source code for implementation and interface of the service available for everything. Further, the freedom and openness determine the flexibility of F/O-Services. We exemplify the freedom and openness exclusively associated with F/O-Services based on possible combinations of derivation (or not) from the source code, modification (or not) of the service interface, and relationship between services (compositional properties) as follows[1]:

1. **Execution Independent Service by Replica of a F/O-Service:** Opening of service allows to create independent services, attributing to the F/O-Service. This is the simplest method enabling the free usage and distribution of a service. Let S_A be a F/O-Service providing a spell checking operation for words, say, $Spell(word)$. Consider S_A provides this service by wrapping PWP spell checker API. Let S_B be an another independent service, providing the same $Spell(word)$, created by replicating the source code of implementation and WSDL of the S_A. Albeit S_A and S_B are performing the same operations, S_A and S_B are two different services, executed separately as in Figure 4.
2. **Execution Dependent Service with Unmodified Interface (irrespective of implementation):** This is a common scenario in SOC as the service's WSDL would

[1] Adapting the styles of [12], we represent the services by the shadowed rectangular boxes. An operation of a service interface is represented as a UML package marked by a stereotype $<< desc >>$. The wrapped application for the service is shown on the left side of the service.

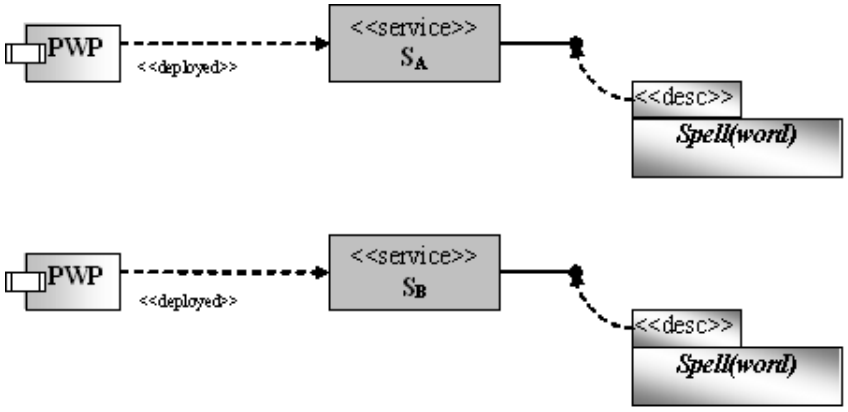

Fig. 4. Replica of a F/O-Service

be obviously available and composition is an indivisible aspect of services. Creating a F/O-Service adds value to a service by distributing the service, not requiring to implement the service again as shown in Figure 5. Let S_B be a service providing a spell checking operation $Spell(word)$ for words, using (copying) the WSDL interface $Spell(word)$ of S_A. S_B is designed in such a way that $Spell(word)$ of S_B directly invokes the operation of S_A, executing on the host of S_A.

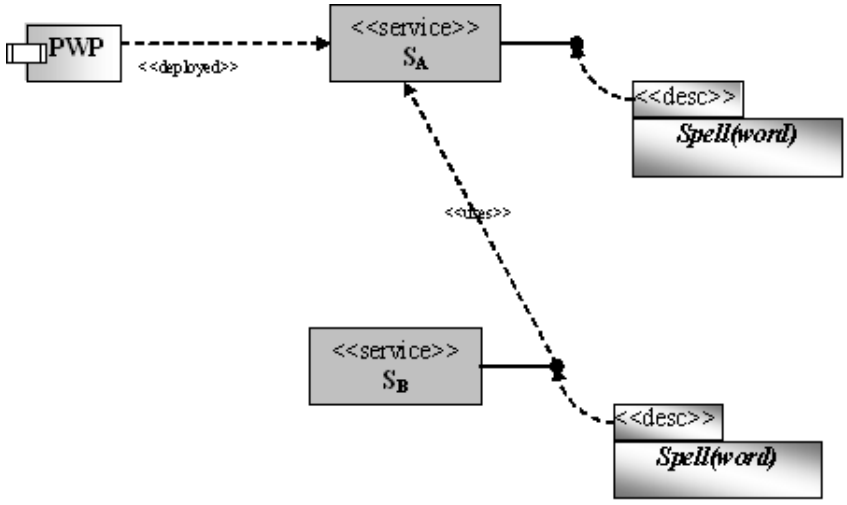

Fig. 5. Service with same interface with composition (implementation not considered)

From a service consumer perspective, in both cases, S_A and S_B are providing exactly the same $Spell(word)$ interface, thus they are interchangeable in an application on the consumer side. The two implementations of S_B are not distinguishable. Theoretically, there will not be any differences in performances of both the services, apart from possible network latency between S_A and S_B.

3. **Execution Independent Service with Unmodified Interface/ Modified Implementation:** An entirely new service could be created from a F/O-Service keeping its interface unchanged and modifying the implementation. Let S_A be a F/O-Service providing $Spell(word)$ by wrapping PWP spell checker API. Let S_B be an another independent service, providing the same $Spell(word)$, created by replicating the WSDL of the S_A. However, S_B provides the operation $Spell(word)$ by wrapping QWP[2] spell check API. Albeit S_A and S_B are performing the same operations, S_A and S_B are two different services, executed separately as in Figure 6. From a service consumer perspective, there could be difference in the performance of S_A and S_B.

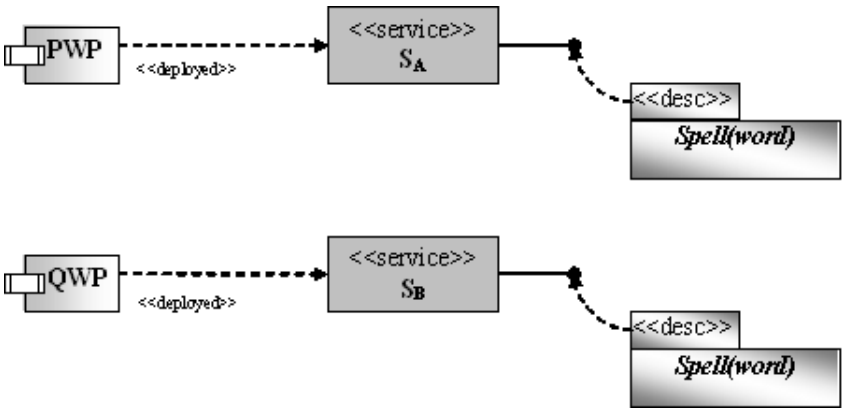

Fig. 6. Independent service with same interface and derived implementation

4. **Execution Independent Service with Modified Interface and Implementation:** Now, consider the case (see Figure 7) similar to $3(a)$ with interface of the F/O-Service S_A be modified in S_B. The interface of S_B provides $Spell(sentence)$ which composes a $parser()$ and repeated invocation of the code derived from S_A, to access PWP API. Now, S_A and S_B are the different services, executing independently. $Spell(sentence)$ of S_B is derived and improved version (having an own additional functionality $parser()$) of $Spell(word)$ of S_A.
5. **Execution Dependent Service with Modified Interface (irrespective of implementation):** Consider a service S_B similar to the case of $3(b)$, but with modified WSDL interface as well as implementation of the open service S_A. $Spell(sentence)$ of S_B comprises a $parser()$ and repeated invocation of the spell checking opera-

[2] QWP is a fictitious name for a proprietary word processor.

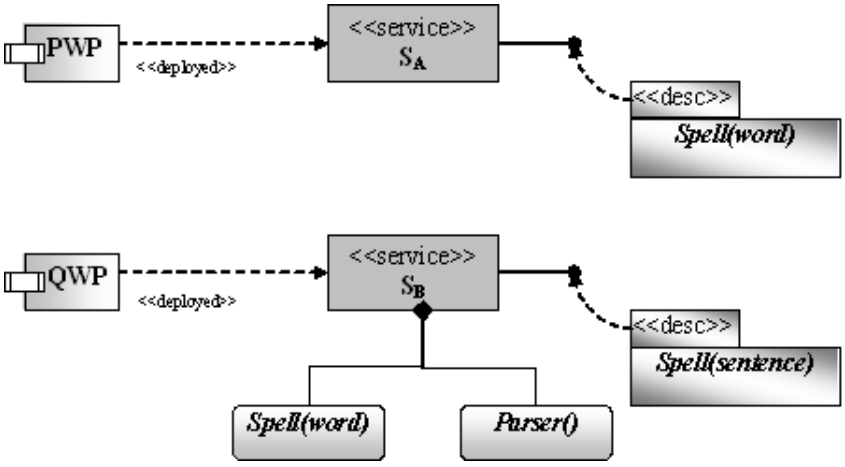

Fig. 7. Replica of a F/O-Service with modified interface and implementation

tion provided by S_A via the interface $Spell(word)$. Thus, the word spell checking operation of S_B is executed in the host of S_A (invoking repetitively the service of $Spell(word)$ of S_A) for spell checking of a given prose $Spell(sentence)$ as in Figure 8.

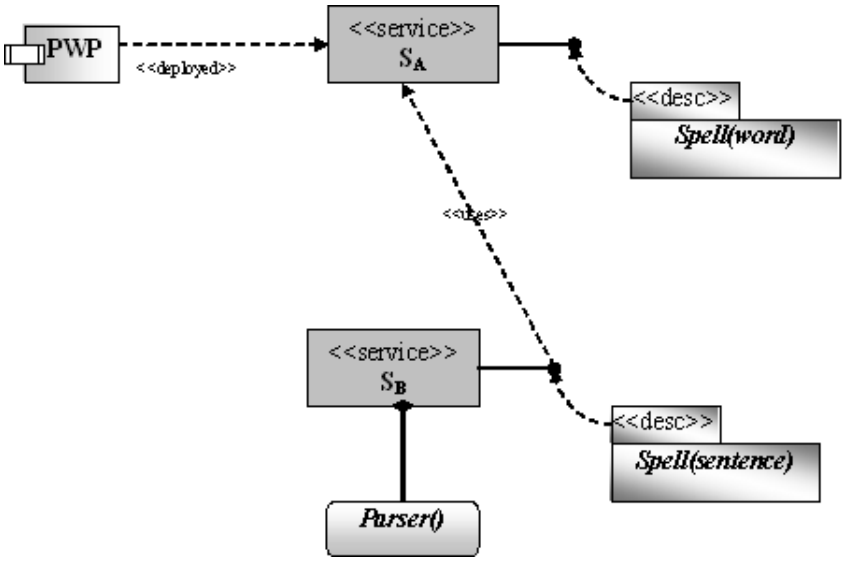

Fig. 8. Service with modified interface, implementation with composition

6. **Execution Independent Service with Modified Interface and Unmodified Implementation:** Let S_A be a F/O-Service providing a spell checking operation for a single word $Spell(word)$, by wrapping PWP spell checker API. Let S_B be an another independent service (see Figure 9), created by replicating the source code of service implementation but modifying the WSDL of S_A, for providing spell checking of multiple words, say, $Spellwords(w1, w2, w3)$.

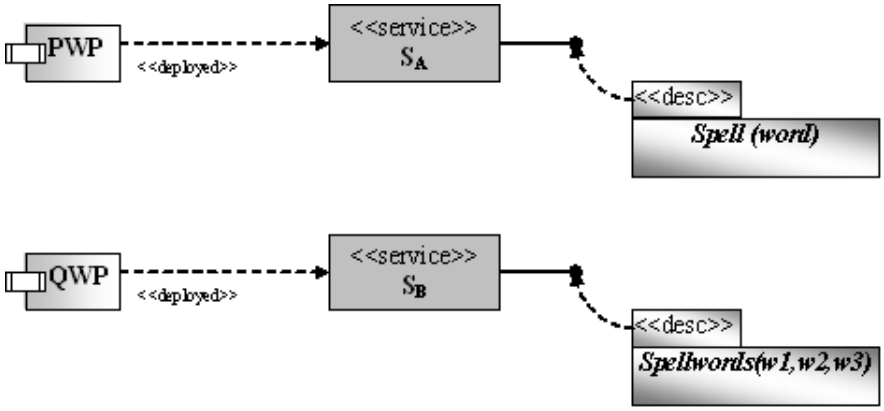

Fig. 9. Replica of a F/O-Service with modified interface and same implementation

Keeping a service interface as not modified (as permitted by a service license) in a dimension, the scenarios illustrated above for service redistribution and service derivation and distribution can be classified as follows:

	Unchanged Implementation	Derived Implementation
No Composition	1	3
Composition	2	

Table 1. Scenarios with unmodified service interfaces

In contrast, if a license allows the modification of a service interface, the scenarios illustrated above for service redistribution and service derivation and distribution can be classified as follows:

	Unchanged Implementation	Derived Implementation
No Composition	6	4
Composition	5	

Table 2. Scenarios with modified service interfaces

6 F/O-Service Business Models

Making F/O-Services may raise an emergent question of how a service provider could profit by providing services. Many FOSS business models are in practice of the community [13]. Some of these business models could be adaptable to the F/O-Service context.

1. ***Support Seller:*** F/O-Services could adopt this scheme where revenue comes from media distribution, branding, training, consulting, and custom development.
2. ***Service Enabler:*** F/O-Services could be created and distributed primarily to support access to revenue-generating on-line services.
3. ***Sell It, Free It:*** Like traditional commercial software, services would begin their product life cycle as closed and then are converted as F/O-Services when appropriate.
4. ***Brand Licensing:*** A F/O-Service provider can charge other service providers/ aggregators/ consumers for the right to use its brand names and trademarks in creating derivative services.

Further, a copyright holder can release his/her works under any license, including multiple licenses and users of that work are allowed use under one of the licenses they choose [14]. Dual licensing is a business model for FOSS exploitation based on the idea of simultaneous use of both FOSS and proprietary licenses [15]. Many well known FOSS projects, including MySQL, Perl, and Qt use dual licensing for their business model. Following the dual licensing strategy, a service can be licensed under FOSS inspired license as well as a proprietary license. In addition to delivering complementary revenue streams, the dual licensing strategy captures a large user base.

7 F/O-Services Development Patterns

FOSS can be viewed as a way of developing software [16] that promotes opening the development process as much as possible, thus offering the possibility to participate in development and distribution to anyone who desires to. The common patterns [16] of FOSS development are:

1. ***Early and Often Releases:*** With a limited set of functionality, the first version of the software is usually released, foreseeing the participation of more co-developers. Further, often releases of the software enables the development of the project in an incremental way.
2. ***Community and Communication:*** A wide community of developers and users geographically dispersed, yet co-ordinated through the Internet, having common goals and interests, make the software to evolve more quickly with high quality reining in errors, by pragmatic code reviews and testing.
3. ***Modularization and Distributed Management:*** Parallel development and code reuse is facilitated by high modularization. The new contributions are added on a source tree (central code repository) of the project. The developers together with the repository manager chose to incorporate strategic decisions.

These basic patterns describe primarily on managing FOSS development, but do not explain analysis, design, and implementation of software engineering methods.

Services are represented as a logical groupings of operations (transactions) having specific business goals [17]. Service as analogous to components, are independent, executable, self-describing and self-contained unit of functionality. However, services are more coarse grained than components. From the view of distributed software development methodologies, the architecture of a system is usually abstracted to a class-level of granularity, which is too low for services modeling. Even in some cases, fine graining of services might be required. As service oriented architecture (SOA) encapsulates stateless, loosely coupled business processes interacting by messages, applying traditional distributed development methodologies to services development would be insufficient. Services are reused in a context not known at design time, envisioning dynamic composition.

As FOSS is developed and maintained by a community of volunteers, anyone can use, modify, redistribute, extend or incorporate it into their own projects adhering the terms of the licenses. Thus, FOSS expands the development of new applications from existing applications. Nowadays, services are being developed by wrapping the existing applications with a thin layer or value added services are created by composition of services. As analogous to FOSS, a new service could be developed by using, modifying or composing with other services. The F/O-Services significantly contributes to the development of new services from existing services by adding new operations. Consider S_X be a F/O-Service. S_Y could be developed by extending S_X in any of the ways discussed in section 5, keeping S_Y as a F/O-Service. S_X could be converted as a new service, S_Z, by incorporating S_Y into S_X, with additional operations.

The early and often releases of services would help in providing functionality rich services. Having more eye balls glancing service interface and implementation, a high level of non-functional properties of services could be attainable.

8 Consequences of Freeing and Open Sourcing of Services

Free services inspired by FOSS licenses could make value addition by composition, resulting composed services as 'free'. Thus, free services (with free licenses) could create a chain effect on composition of services to be free, even if one of the composing service may be not 'free'.

Let S_P and S_Q be the two individual services of a composite service S. S_P and S_Q may be licensed by free or proprietary licenses not imposing restrictions on the use in a composition. The composition of S_P and S_Q inspired by FOSS scheme, is illustrated in Table 3. Making services free will be highly beneficial for government sectors, education, and non-profitable organizations to explore and enjoy the benefits of services.

According to GPL [18], the distribution of GPL'd software must include source code. A GPL'd application delivered as a web service is not actually distributed to the end user. Hence, in this case, the application license does not require to disclose the source code. The nature of web services allows users to interact with the application via an interface, without downloading the software. This can result against the 'free-

S_P	S_Q	$S = \{S_P, S_Q\}$
Free	Free	Free
Free	Proprietary	Free
Proprietary	Proprietary	Free

Table 3. Service composition enriching 'Free Culture'

dom' of GPL, i.e. users consuming services without having access to the source code as delivered by the providers, retaining the rights to modify and distribute.

More precisely, GPL acts on the source code, but not on the use of source code by a service. Consider a service wrapping FWP[3] instead of a proprietary word processor. As FWP is a GPL'd software, a wrapper for FWP is also GPL'd code. However, GPL does not restrict the use of this FWP wrapper provided by a web service. Since, the service is using only the execution of FWP (not the source code of FWP), GPL does not effect the licensing of composite services based on 'FWP wrapper' service. Even the draft version of GPL3 [19] is silent about this issue.

9 Related Work

Today, web services are being used as a component or utility expanding their scope of applications and offer programmatic interfaces to applications [20]. However, many available web services are not even considered as providing relevant business value. Standing on the shoulders of giant industries, the future of web services from the perspective of business is still unclear. Beyond the requirement for standardizing infrastructure and industry standard interfaces of SOA, one of the significant reasons for unfulfilling [21] the promise of web services is the lack of agreement binding service and business. Licensing enables widespread use of services, designing business strategies and relationships. In [22], the author describes a distributed software licensing framework using web services and SOAP. However, [22] addresses a framework using web services but does not address licensing of web services itself. The technical contracts of web services are described in [23], but business and legal contents of contracts are not considered. In [24], we had elaborated the dimensions of web services differing from software and proposed an anatomy of a service license with a set of key negotiation issues.

Leveraging a service oriented paradigm would significantly affect the way people build software systems. However, to achieve this ambitious vision, a substantial development methodology should be in place, comprising specific service-context patterns. Some foundational concepts of service oriented design have already been addressed by [17, 25, 8].

With the growing influence of FOSS initiatives today, it becomes a significant topic to analyze F/O-Services. To the best of our knowledge, the idea of making F/O-Services is completely novel and no previous work exists in this field. Very recently, some informal and unstructured discussions about the concepts of open Services are continuing in

[3] FWP is a fictitious name for a free word processor.

Slashdot [26] and other sites [27]. Much prior to these discussions, we have illustrated the open source based licensing perspectives for the domain of services in [28]. This paper illuminates the concepts of SOC by :

- defining a F/O-Service inspired by FOSS concepts, analyzing the levels of freedom and openness associated with services
- illustrating business models for F/O-Services
- analyzing the development patterns for F/O-Services based on FOSS approach

10 Concluding Remarks

Nowadays, standards are 'open' in SOC. But, the services developed using these standards are unfortunately 'closed'. Hence, we introduced the concept of open services in this paper and analyzed the impacts of freeing and open sourcing services. F/O-Services grant service consumers to add value beyond the concept of composition. The wedding of services with FOSS would be beneficial for both communities, spreading services 'free'ly.

References

1. Feller, J., Fitzgerald, B.: A Framework Analysis of the Open Source Software Development Paradigm. In: Proc. of the 21st Annual International Conference on Information Systems. (2000) 58–69
2. Free Software Foundation: The Free Software Definition. http://www.fsf.org/licensing/essays/free-sw.html (Accessed on Jan. 2006)
3. Brown, A., Booch, G.: Reusing Open Source Software and Practices: The Impact of Open Source on Commercial Vendors. In: Proc. of 7th International Conference on Software Reuse. (2002) 123–136
4. Open Source Initiative: The Open Source Definition. http://opensource.org/docs/definition.php (Accessed on Jan. 2006)
5. Wikipedia: Service. http://en.wikipedia.org/wiki/Services (Accessed on 27.12.2005)
6. Bennett, K., Layzel, P., Budgen, D., Brereton, P., Macaulay, L., Munro, M.: Service-Based Software: The Future for Flexible Software. In: Proceedings of the Asia-Pacific Software Engineering Conference (APSEC). (2000) 214–221
7. Papazoglou, M., Georgakopoulos, D.: Service Oriented Computing. Communications of the ACM **46**(10) (2003) 25–28
8. Ivanyukovich, A., Gangadharan, G.R., D'Andrea, V., Marchese, M.: Towards a Service Oriented Development Methodology. Journal of Integrated Design and Process Science **9**(3) (2005) 53–62
9. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.: Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. Prentice Hall PTR (2005)
10. Alvarez, P.A., Baares, J.A., Ezpeleta, M.J.: Approaching Web Service Coordination and Composition by means of Petri Nets. In: Proceedings of the 3rd International Conference on Service Oriented Computing. (2005) 185–197
11. Dustdar, S., Schreiner, W.: A Survey on Web Services Composition. International Journal of Web and Grid Services **1**(1) (2005) 1–30

12. Heckel, R., Lohmann, M., Thone, S.: Towards a UML Profile for Service Oriented Architectures. In: Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications (MDAFA) . (2003)
13. Raymond, E.: The Magic Cauldron. http://www.catb.org/ esr/writings/magic-cauldron/magic-cauldron.html (1999)
14. Wikipedia: Dual license. http://en.wikipedia.org/wiki/Dual_license (Accessed on 29.12.2005)
15. Valimaki, M.: Dual Licensing in Open Source Software Industry. Systemes d' Information et Management (2003)
16. Robles, G.: A Software Engineering Approach to Libre Software. http://www.opensourcejahrbuch.de/2004/pdfs/III-3-Robles.pdf (2004)
17. Zimmermann, O., Krogdahl, P., Gee, C.: Elements of Service Oriented Analysis and Design. http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/ (2004)
18. Free Software Foundation: GNU General Public License. http://www.gnu.org/copyleft/gpl.html (Accessed on Jan. 2006)
19. Free Software Foundation: GNU General Public License Version 3. http://gplv3.fsf.org (Accessed on Jan. 2006)
20. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services Concepts, Architectures, and Applications. Springer Verlag (2004)
21. Kreger, H.: Fulfilling the Web Service Promise. Communications of the ACM (2003) 29–34
22. Clarke, N.: Distributed Software Licensing Framework based on Web Services and SOAP. http://www.dsg.cs.tcd.ie/∼dowlingj/students/clarken/clarken_02.pdf (May 2002)
23. Tosic, V., Pagurek, B.: On Comprehensive Contractual Descriptions of Web Services. In: Proceedings of the IEEE e-Technology, e-Commerce, and e-Service (EEE). (2005) 444–449
24. D'Andrea, V., Gangadharan, G.R.: Licensing Services: The Rising. In: Proceedings of the IEEE Web Services Based Systems and Applications (ICIW'06), Guadeloupe, French Caribbean. (2006) 142–147
25. Lee, E.: Web Service Implementation Methodology. http://www.oasis-open.org/committees/download.php/13420/fwsi-im-1.0-guidlines-doc-wd-publicReviewDraft.htm (2005)
26. Slashdot: Web Services and Open Source at OSCON. http://developers.slashdot.org/article.pl?sid=06/07/26/1537213 (Posted on July 26, 2006)
27. log.ometer.com: Log for July, 2006. http://log.ometer.com/2006-07.html (Posted on July 29, 2006)
28. D'Andrea, V., Gangadharan, G.R.: Licensing Services: An "Open" Perspective. In: Open Source Systems (IFIP Working Group 2.13 Foundation Conference on Open Source Software), Vol. 203, Springer Verlag. (2006) 143–154